A Supplementary Materials

For easy reference we have summarized the basic algorithm in Pseudocode 1.

```
function $O$ ← Synthesize($I$)
  // $O$: output
  // $I$: input exemplar
  $O_0$ ← Initialize($I$)
  foreach time step $t$
    $O_t$ ← Advect($O_{t-1}$) // application specific advection
    $O_t$ ← Optimize($I$, $O_t$) // compute one frame
  end

function $O_t$ ← Optimize($I$, $O_t$)
  iterate until convergence or enough # of iterations reached
  $\{n(s_i)\}$ ← Search($O_t$, $I$) \// search phase
  Assign($\{n(s_i)\}$, $O_t$) \// assignment phase
    extra solver steps
  return $O_t$

function $\{n(s_i)\}$ ← Search($O_t$, $I$)
  foreach element $s_o$ ∈ $O_t$
    $n(s_o)$ ← output neighborhood around $s_o$
    $n(s_o)$ ← find most similar neighborhood to $n(s_o)$ for $s_i$ ∈ $I$
  end
  return $\{n(s_i)\}$

function Assign($\{n(s_i)\}$, $O_t$)
  foreach output element $s_o$ ∈ $O_t$
    $p(s_o)$ ← least squares from predicted positions
  end

Pseudocode 1: Dynamic element textures for frame-by-frame synthesis.
```

A.1 Algorithm Enhancements

Here we describe enhancements in speed and quality of our basic algorithm.

Factorization  Due to the high dimensionality of the state spaces for dynamic element textures, we cannot expect small input exemplars to provide sufficient coverage for all plausible configurations. Fortunately, we have observed that many phenomena have only loosely coupled spatial geometries and temporal motions. This allows us to perform factorization to enhance synthesis quality. Specifically, given an output neighborhood, we first extract its spatial and temporal parts (the horizontal and vertical portions as illustrated in Figure 3 right), and find the corresponding spatial- and temporal-only best matches in the search step. Both of the two matches provide its own predictions for the relative positions during the assignment step.

Order-independence  Our method can be made order-independent [Lefebvre and Hoppe 2005] in addition to being parallel. That is, it can compute a specific spatial-temporal subset of the output without touching the entire volume, with consistent results regardless of the chosen subset. The key idea, similar to prior order-independent methods, is to gradually expand the footprint from the output subset towards earlier iterations, and synthesize the footprint from earlier to later iterations. Order-independence can be very helpful in a variety of scenarios, such as parallel computing multiple frames of the same animation, or interactive editing for a specific spatial-temporal constraint.